10. Digital Signature Schemes

Digital signature schemes are the electronic equivalent of handwritten signatures, i.e., they allow distinguished principles to sign messages in a way that everybody is able to check the validity of the signatures while additionally ensuring that nobody else is able to produce valid signatures. Technically, digital signatures can thus be seen as an asymmetric variant of MACs: While knowing the secret key is a necessary prerequisite to verify the validity of a MAC, signature schemes enable signature verification based on publicly available keys.

Digital signatures are used in various contexts and applications in practice, e.g., in electronic payment systems and for issuing certificates. In fact, digital signatures might even be of higher importance than encryption schemes in practice, as violations of integrity often turn out to be the most important concern.

10.1 Definition of Digital Signature Schemes

We now formalize the notion of digital signature schemes, or short *signature schemes*. Note that the definition closely resembles the notion of MACs, expressed in an asymmetric scenario.

Definition 10.1 (Signature Schemes) *A* signature scheme *consists of three efficient algorithms* $(\mathsf{Gen}, \mathsf{S}, \mathsf{V})$ *that are defined as follows:*

- *The randomized* key generation algorithm Gen *takes the security parameter n in unary representation as input and returns a pair (pk, sk) of keys, the* public key *pk and the corresponding* private key *sk.*
- *The* signing algorithm S *takes the secret key sk and a message m and returns a tag t. This algorithm may be randomized.*
- *The deterministic* verification algorithm V *takes the public key pk, a message m, and a tag t, and returns* true *or* false.
- *The* message space $\mathcal{M}_{pk}$ *for a public key pk is the set of all m such that $\mathsf{S}(sk, m)$ does not output a distinguished error symbol $\downarrow$ for all sk with $(pk, sk) \in [\mathsf{Gen}(n)]$. We require that, for all $n \in \mathbb{N}$, for all $(pk, sk) \in [\mathsf{Gen}(n)]$ and any message $m \in \mathcal{M}_{pk}$, if $t \leftarrow \mathsf{S}(sk, m)$ then $\mathsf{V}(pk, m, t) = \mathsf{true}$.*

$\diamond$

Most practical signature schemes rely on the message space $\mathcal{M}_{pk} = \{0,1\}^{n_0}$ for a value n_0 that only depends on n but not specifically on pk, or they allow the signing of arbitrary strings, i.e., $\mathcal{M}_{pk} = \{0,1\}^*$.

We again assume that all algorithms (including challengers and adversaries in the following) get the security parameter in unary representation as input so that their efficiency can be meaningfully measured in the security parameter. We do not write this explicitly for the sake of readability.

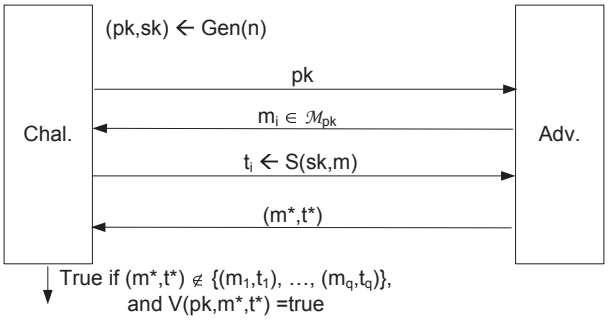

Figure 10.1: CMA-security of Signature Schemes

10.2 CMA-security of Signature Schemes

Security of signature schemes against existential forgery under chosen-message attack (CMA) is then defined similar to the corresponding notion for MACs. The corresponding challenger is called a CMA challenger for signatures, but we simply speak of a CMA challenger if confusion with a CMA challenger for MACs can be excluded from the context.

Definition 10.2 (CMA Challenger for Signatures) *Let* $\mathsf{I} = (\mathsf{Gen}, \mathsf{S}, \mathsf{V})$ *be a signature scheme and let* n *be the security parameter. The* CMA challenger (for signatures) *for* I *and* n *is defined as follows:*

- *It first creates keys* $(pk, sk) \leftarrow \mathsf{Gen}(n)$ *and outputs* pk*.*
- *It then receives messages* $m_i \in \mathcal{M}_{pk}$ *and outputs* $t_i \leftarrow \mathsf{S}(sk, m_i)$*. This stage may repeat arbitrarily but finitely often, thus yielding a sequence of pairs* $(m_1, t_1), \ldots, (m_q, t_q)$*.*
- *Finally, it receives a pair* (m^*, t^*)*. If* t^* *is a valid tag for* m^* *and this pair is not contained in the obtained sequence, i.e., if* $\mathsf{V}(pk, m^*, t^*) = \mathsf{true}$ *and* $\forall i \in \{1, \ldots, q\}\colon (m_i, t_i) \neq (m^*, t^*)$*, then the CMA challenger outputs* true*, and* false *otherwise.*

◇

The game between the CMA challenger and an adversary is depicted in Figure 10.1. The adversary wins the game if it is able to compute a tuple (m^*, t^*) such that the challenger outputs true. This tuple is called an *existential forgery*. Another type of forgery is a *selective forgery*, where an adversary has to be able to find a tag t' *for every* (given) message m'. In the following, $Exp_{\mathsf{A}}^{\mathsf{CMA}}$ denotes the experiment where the adversary A interacts with the CMA challenger, and we let $Exp_{\mathsf{A}}^{\mathsf{CMA}} = \mathsf{true}$ denote the event that the CMA challenger finally outputs true. The advantage can now be defined as a function of n as usual.

Definition 10.3 (CMA Advantage) *Let* $\mathsf{I} = (\mathsf{Gen}, \mathsf{S}, \mathsf{V})$ *be a signature scheme. The advantage of an adversary* A *against the CMA challenger for* I *is defined as follows (as a function of the security parameter* n *again since all algorithms are parametrized with* n*):*

$$Adv^{\mathsf{CMA}}[\mathsf{A}, \mathsf{I}](n) := \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{CMA}} = \mathsf{true}\right].$$

◇

Definition 10.4 (CMA-secure Signature Schemes) *A signature scheme* $\mathsf{I} = (\mathsf{Gen}, \mathsf{S}, \mathsf{V})$ *is* secure against existential forgery under chosen-message attack (CMA) *if* $Adv^{\mathsf{CMA}}[\mathsf{A}, \mathsf{I}](n)$ *is negligible for all efficient adversaries* A*. We often say* CMA-secure signature scheme *for brevity instead of a signature scheme that is secure against existential forgery under chosen-message attack.* $\diamond$

10.3 On Signing Message Digests

Signature schemes are almost always used in conjunction with a hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{D}$, where $\mathcal{D}$ is contained in the set of messages that can be signed. Given a message m, the signature algorithm first applies the hash function yielding $d := \mathsf{H}(m) \in \mathcal{D}$, and then signs this message digest yielding the signature. To verify a signature one computes $d := \mathsf{H}(m) \in \mathcal{D}$ and uses the verification algorithm for this digest.

The hash function used in this design necessarily has to be collision-resistant, as otherwise the following attack can be carried out. First, find a collision for the hash function, i.e., $m \neq m'$ with $\mathsf{H}(m) = \mathsf{H}(m')$, then let the challenger sign m. This signature is a valid signature both for m and m', as their hash-value is the same. Thus both pass verification regardless of the concrete verification algorithm. Note that this attack in practice would typically require that an attacker can come up with two *meaningful* messages that hash to the same value. On the other hand, for MD5 it took roughly one year from finding the first collision to efficiently finding two postscript files that hash to the same value. Thus it is unwise to hope that finding a collision of two meaningful messages is much harder than finding any collision.

Depending on the structure of the verification algorithm the hash function may also need to be one-way. This holds, e.g., for the PKCS#1 signature scheme that we will present below.

10.4 Well-known Signature Schemes

This section is devoted to several well-known signature schemes. We start with so-called one-time signatures which can only be used to sign a single message. While this clearly constitutes a severe constraint in practice, the corresponding schemes were and often still are interesting from a conceptual point of view, and they sometimes are even sufficient for practical applications. After that, we move on to the ElGamal signature scheme and variations thereof, whose security are based on the discrete logarithm problem. Finally, we consider signature schemes based on trapdoor permutations, e.g., based on RSA. This class of signature schemes in particular contains the full-domain hash, which constitutes an efficient scheme that can be proven CMA-secure provided that trapdoor permutations exist; however the proof is in the random oracle model and thus should only be considered a heuristic.

10.4.1 One-time Signatures

We review one of the first one-time signature schemes which is due to Lamport. It is interesting from a theoretical point of view since it shows that secure one-time signature schemes can be achieved only assuming the existence of one-way functions. We further sketch below how this scheme can be extended to allow signing of multiple messages. The drawback of this signature scheme is its bad expansion factor, i.e., the size of the signature compared to the size of the message to be signed.

Lamport's one-time signature scheme works as follows: Let F be a family of keyed one-way functions with key generation algorithm Gen and domains $\mathcal{M}_{pk}$. Let $n \in \mathbb{N}$ denote the security parameter as usual.

- The key generation algorithm creates a public key $pk' \leftarrow \mathsf{Gen}(n)$ of F, randomly chooses $y_{i,j} \leftarrow_{\mathcal{R}} \mathcal{M}_{pk}$, and sets $z_{i,j} := \mathsf{F}(pk', y_{i,j})$ for $1 \leq i \leq n$ and $1 \leq j \leq 2$. The public key pk is then defined as $pk := (pk', (z_{i,j})_{1 \leq i \leq n, 1 \leq j \leq 2})$; the private key sk is defined as $sk := ((y_{i,j})_{1 \leq i \leq n, 1 \leq j \leq 2})$.
- The signature sig of a message $m = m_1 \cdots m_n \in \{0,1\}^n$ with respect to a secret key $sk := ((y_{i,j})_{1 \leq i \leq n, 1 \leq j \leq 2})$ is computed as

$$sig := (y_{1,m_1}, \ldots, y_{n,m_n}).$$

- A signature $sig = (y_{1,m_1}, \ldots, y_{n,m_n})$ on a message $m = m_1 \cdots m_n \in \{0,1\}^n$ is verified with respect to a public key $pk := (pk', (z_{i,j})_{1 \leq i \leq n, 1 \leq j \leq 2})$ by computing

$$\mathsf{V}(pk, m, sig) := \begin{cases} \mathsf{true} & \text{if } \mathsf{F}(pk', y_{i,m_i}) = z_{i,m_i} \text{ for all } 1 \leq i \leq n, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

It can be easily seen that a correctly generated signature passes verification. Let $y_{i,j}$, $z_{i,j}$, and pk' be as constructed by the key generation algorithm, and let $(s_1 \cdots s_n)$ be a signature for the message $(m_1 \cdots m_n)$. But then by construction $s_i = z_{i,m_i}$, and thus by construction of the $z_{i,j}$ we have $s_i = \mathsf{F}(pk', y_{i,m_i})$, thus it passes the verification algorithm.

10.4.2 ElGamal Signatures

The ElGamal signature scheme was proposed in 1985, i.e., roughly at the same time the ElGamal encryption scheme was invented. A modified version of the ElGamal signature scheme was later adopted as the Digital Signature Algorithm (DSA) which we will describe in the next section. Similar to the ElGamal encryption scheme, the ElGamal signature is probabilistic, i.e., signing a message multiple times will yield different signatures. The ElGamal signature scheme is defined as follows:

- The key generation algorithm bears strong similarity to the key generation of the ElGamal encryption scheme: it randomly chooses an n-bit prime p, a generator g of $\mathbb{Z}_p^*$, and an integer $x \in \{1, \ldots, p-1\}$. It then sets $h := g^x$. The public key and secret key are $pk := (p, g, h)$ and $sk := (p, g, x)$, respectively. The message space is $\mathcal{M}_{pk} := \{1, \ldots, p-1\}$.
- A signature sig of a message $m \in \mathcal{M}_{pk}$ with respect to a secret key $sk = (p, g, x)$ is computed by choosing a random $r \leftarrow_{\mathcal{R}} \{1, \ldots, p-1\}$ and by then setting

$$\begin{aligned} s &:= g^r \bmod p, \\ t &:= (m - xs) r^{-1} \bmod (p-1), \\ sig &:= (s, t). \end{aligned}$$

- A signature $sig = (s, t)$ on a message m is verified with respect to a public key $pk = (p, g, h)$ by computing

$$\mathsf{V}(pk, m, (s,t)) := \begin{cases} \mathsf{true} & \text{if } h^s s^t = g^m \bmod p, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

The correctness of ElGamal signatures can be seen as follows: Let $(s,t) \leftarrow \mathsf{S}(sk,m)$ be a signature for pk and sk as defined above. Then we have

$$\begin{aligned}
\mathsf{V}(pk,m,(s,t)) = \mathsf{true} &\Leftrightarrow h^s s^t = g^m \bmod p \\
&\Leftrightarrow g^{xs}(g^r)^{(m-xs)r^{-1}} = g^{xs+r(m-xs)r^{-1}} = g^m \bmod p \\
&\Leftrightarrow xs + (m-xs) = m \bmod p-1 \\
&\Leftrightarrow m = m
\end{aligned}$$

Now we take a look at the security of the ElGamal signature scheme. Let us first note that the security of ElGamal is not proven, and in fact, the ElGamal signature scheme is not CMA-secure in the form it is stated above. We will first give some indication why selectively forging signatures could be as hard as computing discrete logarithms. Suppose Oscar tries to forge a message m without knowing x. If he chooses s first and then tries to find t such that (s,t) is a signature for m, then he has to find $t = \mathsf{DLog}_s g^a h^{-s}$, i.e., to solve a discrete logarithm. If, on the other hand, he chooses t first, then he has to solve the equation $h^s s^t = g^a \bmod p$ for s. For this no feasible solution is known, however, it cannot be reduced to the discrete logarithm problem. There might even be the possibility that one can compute s and t simultaneously; however, no one one has discovered a way to do this so far.

Things change very much if an attacker is satisfied with creating an existential forgery, which we consider as the standard threat for signature schemes we would like to defend against. We will see in the next paragraph that existential forgeries can be computed efficiently for the ElGamal signature scheme.

Existential Forgeries against the ElGamal Signature Scheme The ElGamal signature scheme as described above is existentially forgeable as follows. Suppose we know the public key only, i.e., we know g and h, and we want to find a signature $sig = (s,t)$ which passes the verification algorithm. Suppose i,j are integers such that $1 \leq i,j \leq p-2$, and suppose we express s in the form $s = g^i h^j$. Then the verification condition is $g^m = h^s(g^i h^j)^t \bmod p$, or, equivalently, $g^{m-it} = h^{s+jt} \bmod p$. The later congruence is satisfied if $m - it = 0 \bmod p-1$ and $s + jt = 0 \bmod p-1$. We are given i and j, thus, provided that $\gcd(j, p-1)$, we can easily solve these congruences for t and m, obtaining $t = -sj^{-1}$ and $m = it = -sij^{-1}$. Now from the construction is it immediately clear that (s,t) is a valid signature for m. This attack can be countered by applying a hash function to m before it is input to the ElGamal signature scheme.

10.4.3 Schnorr Signatures

Schnorr signatures have been proposed in 1989 and can be seen as a variant of ElGamal signatures while offering better performance. The ElGamal signature scheme (and its strengthening by using hash functions) rely on a prime p which has to be chosen large enough to prevent an attacker from computing discrete logarithms in $\mathbb{Z}_p^*$. Thus 1024 bit is a good choice; consequently, ElGamal signatures have a length of 2048 bits. Schnorr signatures can significantly decrease this size, typically down to 320 bits by performing computations in subgroups G_q of $\mathbb{Z}_p^*$. (Similar to the definition of ElGamal in subgroups G_q of $\mathbb{Z}_p^*$, we use n to denote the size of q and assume a function $n_p(\cdot)$ such that p is of size $n_p(n)$.)

Let $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_q$ be a hash function. Then the Schnorr signature scheme is defined as follows.

- The key generation algorithm randomly chooses an n-bit prime q and $n_p(n)$-bit prime p such that $q \mid p-1$. It picks an element $g \in \mathbb{Z}_p^*$ of order q and a random $x \leftarrow_{\mathcal{R}} \{1,\ldots,q\}$, and it computes $h := g^x$. The public key and secret key are then defined as $pk = (q,p,g,h)$ and $sk = (q,p,g,x)$, respectively. The message space is $\mathcal{M}_{pk} = \{0,1\}^*$.
- A signature sig on a message $m \in \{0,1\}^*$ with respect to a secret key $sk = (q,p,g,x)$ is computed by choosing a random $r \leftarrow_{\mathcal{R}} \{1,\ldots,q\}$ and by then setting

$$\begin{aligned} s &:= \mathsf{H}(m \parallel g^r), \\ t &:= r + xs \bmod q, \\ sig &:= (s,t). \end{aligned}$$

- A signature $sig = (s,t)$ on a message m is verified with respect to a public key $pk = (q,p,g,h)$ by computing

$$\mathsf{V}(pk,m,(s,t)) := \begin{cases} \mathsf{true}, & \text{if } \mathsf{H}(m \parallel g^t h^{-s}) = s \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

The correctness of Schnorr signatures can be seen as follows: Let $(s,t) \leftarrow \mathsf{S}(sk,m)$ be a signature for pk and sk as defined above. Then we have

$$\begin{aligned} \mathsf{V}(pk,m,(s,t)) = \mathsf{true} &\Leftrightarrow \mathsf{H}(m \parallel g^t h^{-s}) = s \\ &\Leftrightarrow \mathsf{H}(m \parallel g^{r+xs} h^{-s}) = \mathsf{H}(m \parallel g^r) \\ &\Leftrightarrow \mathsf{H}(m \parallel g^{r+xs-xs}) = \mathsf{H}(m \parallel g^r) \\ &\Leftrightarrow \mathsf{H}(m \parallel g^r) = \mathsf{H}(m \parallel g^r) \\ &\Leftrightarrow \mathsf{true} \end{aligned}$$

An attacker that can compute discrete logarithms can clearly forge Schnorr signatures. The converse is, however, not known.

10.4.4 Digital Signatures Algorithm (DSA)

The digital signature algorithm (DSA) was proposed by the NIST in 1991 for the Digital Signature Standard (DSS). Some criticisms however arose when the standard was proposed: One (yet minor) issue was the "closed-door" approach that was employed in the process which did not involve U.S. industry. The more severe point of criticism was however that the size of the modulus p was initially fixed to 512 bits. The standard was later updated to allow moduli p of larger size, and in 2001 the NIST itself recommended to use 1024 bit primes p.

The Digital Signature Algorithm is defined as follows:

- The key generation algorithm randomly chooses an L-bit prime p where L has to satisfy $L = 0 \bmod 64$ and $512 \leq L \leq 1024$, and a 160-bit prime q such that $q \mid p-1$. It then randomly chooses $g \in \mathbb{Z}_p^*$ of order q and $x \leftarrow_{\mathcal{R}} \{0,\ldots,q-1\}$. Finally it sets $h := g^x$. The public key and secret key are then defined as $pk = (q,p,g,h)$ $sk = (q,p,g,x)$, respectively.

- A signature sig on a message $m \in \{0,1\}^*$ with respect to a secret key $sk = (q,p,g,x)$ is computed by choosing a random $r \leftarrow_{\mathcal{R}} \{1,\ldots,q\}$ and by then setting

$$\begin{aligned} s &:= (g^r \bmod p) \bmod q, \\ t &:= (\mathsf{SHA\text{-}1}(m) + xs) r^{-1} \bmod q, \\ sig &:= (s,t), \end{aligned}$$

where SHA-1 denotes the respective hash function. If either s or t equals 0, a new r is chosen and the signature is recomputed. (This happens only with very small probability and thus does not raise a problem in practice).

- A signature $sig = (s, t)$ on a message m is verified with respect to a public key $pk = (q, p, g, h)$ by computing

$$\mathsf{V}(pk, m, (s,t)) := \begin{cases} \mathsf{true} & \text{if } (g^{\mathsf{SHA\text{-}1}(m)t^{-1} \bmod q} h^{st^{-1} \bmod q} \bmod p) \bmod q = s, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Correctness can be shown as usual.

10.4.5 RSA-based Signature Schemes

In this section we describe a very simple construction of a signature scheme based on the RSA trapdoor permutation. This construction can be found in various books, however, it can easily be broken by various attacks. Later we will see how these attacks can be countered by using hash functions and including redundancy inside the signature, as it is done for example in the PKCS#1 standard.

The naive RSA signature scheme works as follows:

- Keys are generated exactly as for (naive) RSA encryption: Randomly choose two n-bit primes p and q, let $N := pq$, and pick e, d such that $ed = 1 \bmod \varphi(N)$. Let $pk = (N, e)$ be the public key, and let $sk = d$ be the secret key.
- To sign a message $m \in \{1, \ldots, N-1\}$ with respect to the secret key $sk = d$ one computes

$$\mathsf{S}(sk, m) := m^d \bmod N.$$

- A signature sig of a message m with respect to the public key pk is verified by computing

$$\mathsf{V}(pk, m, s) := \begin{cases} \mathsf{true} & \text{if } sig^e = m \bmod N, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Correct verification is a direct consequence of Lemma 9.1.

Attacks against Naive RSA Signatures There are several attacks against the above signature scheme. The first attack is an existential forgery under a passive attack: One starts by picking an arbitrary $sig \in \mathbb{Z}_N$ and computes $m := sig^e \bmod N$. Then sig is a signature for the message m, as one can easily verify.

A more sophisticated attack is the following, which can find selective forgeries under an active attack. It uses blinding techniques similar to those we have seen earlier. Suppose the adversary wants to get a signature for a message m. He chooses a random $r \in \mathbb{Z}_N^*$ and computes $m' := m \cdot r^e \bmod N$. He asks the signer to sign m', yielding a signature $s' = (m')^d \bmod N$. Finally we compute $s := s'/r$. Then s is a valid signature for m, as the following computation shows:

$$s^e = \left(\frac{s'}{r}\right)^e = \left(\frac{(m')^d}{r}\right)^e = \left(\frac{(mr^e)^d}{r}\right)^e = \left(\frac{m^d \cdot r}{r}\right)^e = \left(m^d\right)^e = m \bmod N.$$

Note that the ability to let a message be signed blindly can even be a desirable feature which is used in a cryptographic primitive called *blind signature*. This is used whenever it is desirable that the signer should not learn what he is actually signing, e.g., this preserves anonymity in electronic payment systems. We will see this in more detail later in the course.

Adding Redundancy One possibility to counter these attacks is to add redundancy to the message before applying the RSA signature operation, i.e., one computes $sig := (\mathsf{red}(m), m)^d \bmod N$, and verifies by testing $sig^e = (\mathsf{red}(m), m) \bmod N$. Now what should this padding look like? A bad way to do it is to prepend zeroes, as this enables mainly the same attacks as before. A good thing is to prepend something "chaotic", e.g., the encryption of the message m under DES with a fixed (public) key. Then a signature needs to be rejected if the padding is not of the correct form. This padding is used in an ISO standard. The padding in the PKCS#1 standard is similar: There a fixed prefix of the following form is prepended to the message (in hexadecimal notation)

$$0001\ 0101\ 0101\ \ldots\ 0101\ 0000\ ||\ \mathsf{H}(m).$$

10.4.6 Full-domain Hash

The full-domain hash (FDH) signature scheme, or short full-domain hash, uses the common design principle *hash-then-sign*, cf. Section 10.3. It is defined over an arbitrary trapdoor permutation and an arbitrary hash function. The only requirement is that it is important that the range of the hash function is the same as the domain of the trapdoor permutation (thus the name "full-domain").

Let a family of keyed trapdoor permutations F with key generation Gen and domains $\mathcal{M}_{pk}$ be given. Furthermore, let $\mathsf{H}\colon \{0,1\}^* \to \mathcal{M}_{pk}$ be a hash function. Then the full-domain hash (FDH) signature scheme is defined as follows:

- The key generation algorithm simply generates the key for the underlying trapdoor permutation: Let $(pk, sk) \leftarrow \mathsf{Gen}(n)$; the public and the secret key of the signature scheme are then pk and sk, respectively.
- The signature sig of a message $m \in \mathcal{M}_{pk}$ with respect to the secret key sk is computed as

$$sig := \mathsf{F}^{-1}(sk, \mathsf{H}(m)).$$

- A signature sig on a message m is verified with respect to a public key pk by computing

$$\mathsf{V}(pk, m, sig) := \begin{cases} \mathsf{true} & \text{if } \mathsf{F}(pk, sig) = \mathsf{H}(m), \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Theorem 10.1 (Bellare, Rogaway'94) *The FDH signature scheme is existentially unforgeable under chosen-message attack assuming that F is one-way. This holds in the random oracle model, i.e., H is considered an ideal hash functions.* □

Proof. Given an adversary A that breaks the FDH signature scheme with not negligible probability, we construct an adversary B that inverts the trapdoor permutation F with not negligible probability.

Assume the attacker A makes a polynomially bounded number of q_{hash} queries $a_1, \ldots, a_{q_{hash}}$ to the random oracle and a polynomially bounded number of q_{sign} queries $m_1, \ldots, m_{q_{sign}}$ to the signature challenger and finally outputs a candidate forgery (m, sig). Without loss of generality we may assume that a message m_j was queried to the random oracle before it is queried to the signature oracle, and similarly, that the forged message m was queried to the random oracle. (Otherwise we can construct an adversary A' which behaves identical as A, but additionally makes these queries, and this A' still breaks the CMA-security of FDH.)

In the first step the adversary B gets the public key pk and a value $y = \mathsf{F}(x)$ for some unknown, randomly chosen value $x \in \mathcal{M}_{pk}$. Subsequently he does the following:

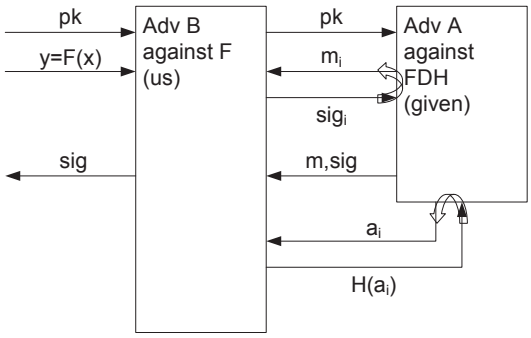

Figure 10.2: Attack game against the FDH

- It first chooses a random $k \leftarrow_{\mathcal{R}} \{1, \ldots, q_{hash} + q_{sign} + 1\}$.
- Upon receiving any hash query a_j, it tests if this value was queried before, in which case it answers the same value as before (all queries and the corresponding results are stored by B). Otherwise, if $j = k$ then it returns y, else it chooses a random $s_j \leftarrow_{\mathcal{R}} \mathcal{M}_{pk}$ and returns $\mathsf{F}(pk, s_j) =: h_j$ (Note that h_i is random because s_i is random and F is a permutation).
- Upon receiving a signature query m_l, it determines the first hash query a_j for this m_l (we assumed that m_l was queried to the random oracle before, so such an a_j will exist). If $j = k$ then output $\downarrow$, else output s_j.
- Receiving the (candidate) forgery (m, sig) find the first hash query a_j for m (again, we assumed this always exists). If $j = k$ then output sig, else abort with $\downarrow$.

Note first that in order for (m, sig) being a forgery, we must have $m \neq m_i$ for all remaining messages m_i that occurred in sign queries: Since signing is deterministic in FDH, this would imply $sig = sig_i$ so that the CMA challenger would output false upon receiving (m, sig) as a candidate forgery. Let $(a_1, \ldots, a_{q_{hash}+q_{sign}+1})$ denote the sequence of pairwise different messages that were used as hash queries (also containing the messages that we would have to add if we replaced A by A'). Then the adversary B gives a perfectly correct simulation to A provided that the two following conditions are satisfied. First, for all messages m_i that were used as sign queries, the corresponding hash queries $a_j = m_i$ satisfies $j \neq k$. Secondly, we have $a_k = m$. Since $m \neq m_i$ as discussed above, since k is randomly chosen from $\{1, \ldots, q_{hash} + q_{sign} + 1\}$, and since $y = \mathsf{F}(pk, x)$ is random because x is random and F is a permutation, these two conditions are simultaneously satisfied with probability at least $\frac{1}{q_{hash}+q_{sign}+1}$. If A succeeds in forging the signature for m in this case, then he outputs a signature sig such that $\mathsf{F}(pk, sig) = \mathsf{H}(m)$; thus sig is indeed a pre-image of y by construction. One finally obtains

$$\begin{aligned} \mathsf{Pr}[\mathsf{A} \text{ wins}] &= \mathsf{Pr}[\mathsf{F}(sig) = \mathsf{F}(x);\ x \leftarrow_{\mathcal{R}} \{0,1\}^n,\ y := \mathsf{F}(x),\ sig \leftarrow \mathsf{B}(n, y)] \\ &\geq \frac{1}{q_{hash} + q_{sign} + 1} \cdot Adv^{\mathsf{CMA}}[\mathsf{A}, FDH]. \end{aligned}$$

■